동적 제어 정보를 이용한 효율적인 프로그램 슬라이싱 알고리즘

박순형[†] · 정은이[††] · 박만곤[†††]

요 약

일반적인 소프트웨어 시스템은 새로운 요구와 오류의 발견으로 인해 계속적인 개발과 확장 그리고 수정이 요구된다. 이러한 일련의 작업 과정에서 기존 프로그램의 정확한 이해는 매우 필요하다. 어떤 프로그램의 특정 명령문에 있는 변수에 대한 관련 명령문을 찾고 싶을 때 프로그래머는 입력 자료의 값에 대해 프로그램의 실행궤도 추적을 통해 프로그램을 분석한다. 그러므로, 현재 입력 값에 영향을 끼치는 모든 명령문들에 관련된 동적 프로그램 슬라이싱(dynamic program slicing)과 이를 구현하는 기술의 개발은 매우 중요하다고 할 것이다. 그러나 전통적인 동적 슬라이싱 기법은 구현 초기에 프로그램 실행이력을 만들어야하는 불편이 있었다. 본 논문에서는 실행이력 파일을 사용하지 않고 동적 제어 정보와 프로그램 슬라이싱 기법을 사용하여 효율적으로 프로그램 슬라이스를 산출하는 알고리즘을 제시하였고 이것을 프로그래밍 한 뒤 예제 프로그램을 적용시켜 구현하였다. 그리고, 본 논문에서 제시한 슬라이스 생성기법이 기존의 기법 보다 더 효율성이 높다는 것을 보였다.

An Efficient Program Slicing Algorithm using Dynamic Control Information

Soon-Hyung Park[†], Eun-Yi Jung[††] and Man-Gon Park[†††]

ABSTRACT

For the operation of the practical software systems, the development of new software, extension and modification phases of current software are successively needed through the new requirements added and their errors debugging detected. We need to understand current program exactly during a working serial jobs. When we'd like to extract the statements which influence the variable of specific statement of program, we generally analyze the program behavior through execution trace of program for the input values. It is important to compute dynamic program slice related to all statements that actually affect the value of a variable occurrence for a given program input and to develop techniques of its implementation. But traditional dynamic slicing techniques are inconvenient to make program execution history at the beginning implementation. In this paper, we propose a new improved algorithm which can produce program slice by use of dynamic control information and program slicing techniques except execution history file. Also we can find that the proposed program slicing approach is more efficient than the traditional program slicing algorithm on the practical testing environment.

1. 서 론

프로그램 슬라이싱은 기준 노드의 특정 변수 var의 값에 직 간접적으로 영향을 끼치는 프로그램 P에 있는 모든 명령문들을 찾는 것이다[1,4,10,11]. 프로그램 유지보수 작업의 일환으로 현행 소프트웨어 시스템에 대해 지속적으로 프로그램을 개발하고 확장할 때 기존의 컴포넌트를 재사용 하거나 효율적으로 프로그램을 테스팅 할 수 있는 기법을 이용할 수 있

† 정회원 : 동의공업대학 전자계산과 교수
†† 정회원 : 춘해대학 멀티미디어정보과 교수
††† 종신회원 : 부경대학교 컴퓨터멀티미디어공학부 교수

다면 소프트웨어의 생산성과 품질은 매우 향상될 것이다. 이를 위해 기존의 프로그램에 대해 정확한 분석과 이해는 매우 필요할 것이다. 그러므로 필요로하는 부분들을 정확하고 효율적으로 프로그램을 슬라이싱하는 기법의 개발은 매우 필요하다[2,3].

정적 슬라이스(static slice)는 주어진 변수 즉, 슬라이싱 기준에 영향을 주는 모든 명령문의 집합이고, 동적 슬라이스(dynamic slice)는 프로그램의 입력 자료의 값에 의해 발생되는 프로그램의 실행 궤도를 추적한 다음 추적궤도 상에서 기준 변수에 실제적으로 영향을 주는 모든 명령문들로 구성되어 있다[12, 14,15].

프로그램에서 특정 명령문 S이 있는 변수 var의 값을 디버깅하거나 테스팅을 하기 위하여 프로그래머는 입력자료의 값이나 초기정의문을 통해 해당 변수에 변화를 끼치는 변수들의 값을 추적함으로써 프로그램을 분석한다. 그러므로 현재의 입력 값과 기준 명령문 노드의 변수에 영향을 끼치는 모든 명령문들을 찾는 동적 슬라이싱(dynamic slicing)이 입력 파일의 값과는 무관한 정적 슬라이싱(static slicing)보다 프로그램의 테스팅과 디버깅에 더 적합하다고 할 것이다.

그러나, 전통적인 동적 슬라이싱 기법은 입력파일에 의한 프로그램 추적작업을 통해 실행이력 파일을 만드는 작업이 먼저 선행되어야 한다. 그리고, 이러한 과정은 동적 슬라이싱의 실행 효율을 저하시키므로 본 논문에서는 실행이력 파일을 만들지 않고 기존의 기법 보다 효율성이 뛰어난 슬라이싱 기법을 제안하였다.

2장에서는 기존의 슬라이싱 기법에 대해 설명하였으며 3장에는 본 논문에서 제시한 동적 제어 정보를 이용한 효과적인 프로그램 슬라이싱 알고리즘을 제시하였고 프로그래밍 언어를 사용하여 실제 구현한 결과를 프로그램 종속 그래프(program dependence graph)를 통해 나타내었다. 4장에서는 본 논문에서 제시한 효율적인 프로그램 슬라이싱 알고리즘이 기존 알고리즘에 비해 효율적임을 비교 고찰하였다.

2. 관련 프로그램 슬라이싱 기법

2.1 PDG를 이용한 정적 슬라이싱

어떤 한 프로그램에 대해 자료 흐름 분석과 제어 흐름 분석을 통해 프로그램 종속 그래프로 나타낼 수 있다. 프로그램 종속 그래프는 각각의 명령문 에 대해 한 개의 노드를 가진다. 노드들은 간선(edge)들로 연결되며 간선에는 두 가지 종류 즉, 자료 종속 간선과 제어 종속 간선이 있다[7,8].

정점(vertex) v_i으로부터 정점 v_j 까지 자료 종속 간선은 정점 v_i에서 실행된 결과는 정점 v_j에서 실행된 값에 직접 종속된다. 즉, 정점 v_i 의 실행 결과를 구하기 위해 v_j에서 정의되어진 변수 var을 사용한다. 그리고, v_i으로부터 정점 v_j 까지 제어 종속 간선은 노드 v_i가 노드 v_j에서 술어 표현의 boolean 결과에 따라 실행할 수도 안 할 수도 있다는 의미이다.

정적 슬라이스는 노드 n에서 var에 도달하는 모든 것을 찾은 후 그 노드에서 시작하는 프로그램 종속 그래프를 운행(traversing) 함으로써 쉽게 구성될 수 있다. 따라서, (그림 1)의 예제 프로그램 1에 있는 명령문 10의 변수 Y에 대한 정적 슬라이스는 {1, 2, 3, 5, 6, 8}이 된다.

```
          begin
S1:           read(X)
S2:           if (X < 0)
              then
S3:               Y := f1(X);
S4:               Z := g1(X);
              else
S5:               if (X = 0)
                  then
S6:                   Y := f2(X);
S7:                   Z := g2(X);
                  else
S8:                   Y := f3(X);
S9:                   Z := g3(X);
                  end_if;
              end_if;
S10:          write(Y);
S11:          write(Z);
          end.
```

그림 1. 예제 프로그램 1
Fig. 1. Example Program 1

2.2 RDDG를 이용한 동적 슬라이싱

동적 슬라이스는 주어진 프로그램의 입력 자료에 대해 어떤 실행위치 q에서의 기준 변수와 관련된 프로그램의 부분 집합이며 동적 슬라이스는 원래의 프로그램과 실행 결과가 일치하여야 한다[5,6]. 그러므로 동적 슬라이싱을 하기 전에 프로그램 입력 x에

대한 실행이력(H_x) 과 프로그램 P의 동적 슬라이싱 기준 C = (x, I^q, V)을 먼저 정의해야 한다. 실행 이력이란 주어진 시험사례를 실행하는 동안 방문된 순서에 의한 일련의 정점들인 {v_1, v_2, ... , v_n}의 집합이다[11]. I^p는 H_x 내에 있는 position q에서의 명령문 I (즉, $H_x(p)$ = I)이다. 그리고, V는 I^p에서의 기준 변수이다.

슬라이싱 기준 C에 대한 프로그램 P의 동적 슬라이스는 0 개 이상의 명령문을 삭제함으로써 P로 부터 얻어지는 정확하고 실행 가능한 프로그램 P'이다[7-9].

주어진 기준 변수에 대한 동적 슬라이스를 구하기 위한 기법은 여러 가지 있지만, 특히 반복type 노드들에 대한 효율적인 동적 슬라이싱을 위해 축소 동적 종속 그래프 기법을 제안한 Agrawal & Horgan이 제안한 기법을 소개한다[1,5,6].

① 그래프에서 모든 노드들을 시작 시점에서 점선으로 그린다.

② 명령문들이 실행됨에 따라 발생하는 새로운 종속에 대응하는 간선들을 실선으로 만든다. 실행 이력에서 어떤 한 명령문의 각 발생들에 대해서 해당 명령문에 종속성 간선을 가진 단지 한 개의 명령문이면서 동일한 종속성을 가진 또 다른 노드가 존재하지 않을 경우 새로운 노드를 만든다.

③ 실선으로 표시된 간선을 따라 운행하고 도달된 노드들을 굵은 실선으로 표현하고 슬라이스에 포함시킨다.

(그림 2)의 예제 프로그램 2의 시험 사례로서 N = 3 이고 X = {-2, 4, -3} 일 때 S9의 변수 Z에 대한 동적 슬라이스를 산출한다. 이 때, 실행 이력은 {1^1, 2^1, 3^1, 4^1, 5^1, 6^1, 8^1, 9^1, 10^1, 3^2, 4^2, 5^2, 7^1, 8^2, 9^2, 10^2, 3^3, 4^3, 5^3, 6^2, 8^3, 9^3, 10^3, 3^4}이 된다. 위의 기법들을 적용시킨 결과는 (그림 3)에 축소동적종속그래프(RDDG : Reduced Dynamic Dependence Graph)로 나타내었으며, 굵은 원은 슬라이스된 노드들이다.

```
        begin
S1:         read(N)
S2:         I := 1;
S3:         while (I <= N)
          do
S4:             read(X)
S5:             if (X < 0)
                then
S6:                 Y := f1(X);
                else
S7:                 Y := f2(X);
                end_if;
S8:             Z := f3(Y);
S9:             write(Z);
S10:            I := I + 1;
            end_while;
        end.
```

그림 2. 예제 프로그램 2
Fig. 2. Example Program 2

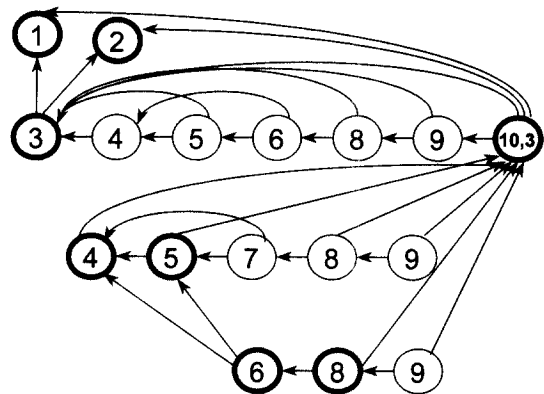
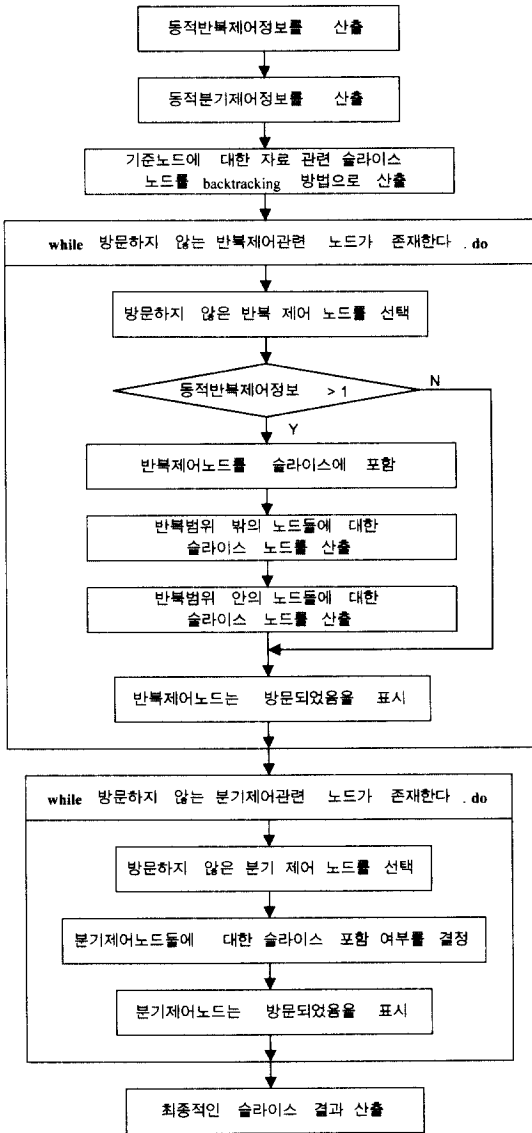



그림 3. 예제 프로그램 2의 축소 동적 종속 그래프
Fig. 3. RDDG of Example Program 2

3. 동적 제어 정보를 이용한 효율적인 프로그램 슬라이싱 기법

본 논문에서 제시한 프로그램 슬라이싱 기법은 입력 자료를 이용한 동적 제어 정보를 산출하는 작업부터 시작한다. 프로그램을 구성하는 모든 노드들은 자료 종속 노드와 제어 종속 노드에 소속되어 있으며 제어 종속 노드는 반복 제어 종속 노드와 분기 제어 종속 노드로 구분된다. 본 논문에서 제시한 효율적인 슬라이싱 기법의 처리 순서도는 다음과 같다.

(1) 동적 반복 제어정보를 산출

동적반복제어정보 테이블에는 해당 제어 술부의 값을 계산하여 반복회수에 대한 정보를 저장한다.

(2) 동적 분기 제어정보를 산출
분기별 반복 카운터에 대한 정보를 산출하여 동적분기제어정보 테이블에 저장한다.

(3) 자료관련 노드에 대한 슬라이싱
동적반복제어정보 테이블과 동적분기정보 테이블에 저장된 정보를 이용하여 주어진 기준노드에 대한 자료관련 슬라이스 노드를 backtracking 방법으로 산출한다.

(4) 반복제어 관련 노드에 대한 슬라이싱
① 반복제어 노드를 중심으로 반복범위 안의 노드들에 대한 슬라이싱을 산출한다.
② 반복제어 노드를 중심으로 반복범위 밖의 노드들에 대한 슬라이싱을 산출한다.

(5) 분기제어 관련 노드들에 대한 슬라이싱
(3)의 처리 결과 분기별 노드 그룹 중 1 개 노드 이상이 각각 슬라이스에 포함되었는지를 체크하여 포함되었으면 분기제어 노드를 슬라이스에 포함한다.

(6) 최종 슬라이스 산출
(3), (4) 그리고 (5)의 슬라이스 산출 결과를 바탕으로 최종 슬라이스를 산출한다.

본 논문에서는 효율적인 프로그램 슬라이싱을 위한 알고리즘을 제시하고 이 알고리즘을 프로그래밍 언어를 통해 실제 구현하였다. 그리고 그 결과를 프로그램 종속 그래프로 나타내었다.

본 논문에서 프로그래밍 언어의 구조는 프로그래밍 언어 ML에서 유사 구조로 채택된 let-in구조를 사용한다[13].

let <declarations> in <expression>

3.1 효율적인 프로그램 슬라이싱을 위한 입력 자료

프로그램 슬라이싱 알고리즘을 구현하기 위해 노드 관련 자료가 필요하다. 노드 관련 자료란 각 명령문에 해당되는 노드들의 구성 요소를 저장해 놓은 자료를 말하며 노드 번호, 노드 type, DEF, 그리고 USE로 구성된다. 노드 type에는 입력, 서술, 분기, 반복, 출력 등이 있으며, DEF(n)은 노드 n에서 바뀌어진 값을 가진 변수의 집합이고, USE(n)은 노드 n에서 사용된 값을 가진 변수의 집합이다.

3.2 효율적인 프로그램 슬라이싱 알고리즘

본 논문에서 제시한 프로그램 슬라이싱 알고리즘에는 두 개의 기준 변수 테이블과 한 개의 초기 정의용 변수 테이블이 운영된다. 두 개의 기준 변수 테이블이란 서술 기준 변수 테이블(*DefnBaseNode*)과 제어 기준 변수 테이블(*CntlBaseNode*)을 말하는데 서술 기준 변수 테이블은 직전 서술 명령문의 USE에 관련된 변수의 집합이고, 제어 기준 변수 테이블은 직전 제어 관련 명령문의 USE에 관련된 변수로써 슬라이싱 제어 기준이 되는 변수들의 집합을 말한다.

본 논문에서 제시한 효율적으로 프로그램 슬라이스(*ReachableStmts*)를 산출하는 알고리즘은 다음과 같다.

```
ReducedDynamicDep(<prevhist | next>)=
   let ReducedDynamicDep( prevhist )
       = ( V, A, ReachableStmts, DefnBaseNode,
           CntlBaseNode ),
     D = ∪var← use(v) DefnBaseNode( var ),
     C = ∪x← use(v) CntlBaseNode ( x )
   in if InitSw = 0 then
        SearchBaseVar( BaseVar, ReachableStmt,
                Def ),
        InitSw = InitSw + 1
     end if
     AssgnProc( NodeType, Def, Use,
                DefBaseNode ),
     ControlProc( NodeType, Def,
                DefnBaseNode, CntlBaseNode )

SearchBaseVar( BaseVar, ReachableStmt, Def ) =
   let R = { v } ∪ ∪x∈D ReachableStmts( x )
   in if BaseVar = Def( Var ) then R,
           AddDefnNode( DefnBaseNode, Use )
     else SearchBaseVar( BaseVar,
               ReachableStmt, Def( next ))
     end if

AssgnProc( NodeType, Def, Use,
                DefnBaseNode ) =
   let R = { v } ∪ ∪x∈D ReachableStmts( x ),
   in if NodeType( x ) = DefnType then
        if Def( x ) ∈ D then
           DelDefnNode( DefnBaseNode, Def )
           if Use( x ) not = null then
              AddDefnNode(DefnBaseNode, Use)
           end if
         end if
       end if

ControlProc( NodeType, Def, DefnBaseNode,
           CntlBaseNode ) =
   let R = { v } ∪ ∪x∈D ReachableStmts( x )
   in MakeReptNode( NodeType, Use,
                    CntlBaseNode ),
     MakeBrchNode( DynBrchInfo )

MakeReptNode( NodeType, Use,
                    CntlBaseNode ) =
   let R = { v } ∪ ∪x∈D ReachableStmts( x )
   in if NodeType( x ) = 'Repeat' then
        if ( RepCnt > 1 )  then R
          AddCntlNode( CntlBaseNode, Use )
          if Def( x ) ∈ C then
            DelCntlNode( CntlBaseNode, Def )
            if Use( x ) not = null then
              AddCntlNode(CntlBaseNode, Use)
          end if
         end if
      end if
    end if

MakeBrchNode( DynBrchInfo )
   let R = { v } ∪ ∪x∈D ReachableStmts( x )
   in if DynBrchInfo( n ) ∈ ReachableStmts( x )
       then R

AddDefnNode( DefnBaseNode, Var ) =
   ∪x∈D Var( x ) ∪ ∪x∈D DefnBaseNode( x )

DelDefnNode( DefnBaseNode, Var ) =
   ∪x∈D DefnBaseNode( x ) - ∪x∈D Var( x )

AddCntlNode( CntlBaseNode, Var ) =
   ∪x∈D Var( x ) ∪ ∪x∈D CntlBaseNode( x )
```

3.3 적용 사례

(그림 2)의 예제 프로그램 2에 본 논문에서 제시한 프로그램 슬라이싱 알고리즘을 적용시켜 보자. 시험 사례는 N = 3이고 X = {-2, 4, -3} 그리고, 슬라이싱 기준은 S10의 변수 Z라고 하자.

(1) 동적 반복 제어정보를 산출한다.
  S3에서 동적 반복 제어정보 RCI(3) = {3}

(2) 동적 분기 제어정보를 산출한다.
  S5에서 동적 분기 제어정보 BCI(6) = {1, 3}, BCI(7) = {2}

(3) 기준노드에 대한 자료 관련 슬라이스 노드를 backtracking 방법으로 산출한다.
⇒ {S4, S6, S8, S9}

실행순서	노드번호	*DefnBaseNode*
1	S9	Z
2	S8	Y
3	S6	X
4	(S5)	X
5	S4	blank

(4) 반복노드에 대한 슬라이스 노드를 backtracking 방법으로 산출한다.
⇒ {S1, S2, S3, S10}

(5) 분기 노드의 슬라이스 포함여부를 체크한다. (3)의 슬라이스 결과 분기 그룹 중 슬라이스에 포함되지 않는 분기 그룹이 존재함으로 분기 제어 노드 S5는 슬라이스에 포함되지 않는다.

(6) 최종적인 슬라이스 결과를 산출한다.
⇒ {S1, S2, S3, S4, S6, S8, S9, S10}

노드 관련 자료는 <표 1>에 나타나 있다. 본 논문에서 제시한 프로그램 슬라이싱 알고리즘을 적용시킨 결과를 프로그램 종속 그래프로 표현하면 (그림 4)와 같다. 노드는 자료종속 노드와 제어종속 노드로 나누며, 제어종속 노드는 반복노드와 분기노드로 나눈다. 자료종속 노드는 점선 원으로 표시하며, 슬라이스에 포함되면 실선 원으로 표시된다. 반복대상 노드는 점선 사각형으로 표시하며 반복그룹 내에 입력문이 있거나 자료슬라이싱을 구할 때 영향을 끼치게 되면 반복노드는 실선 사각형으로 표시되고 슬라이스에 포함된다. 그리고, 분기노드는 육각형으로 표시하며 분기되는 두 개의 노드그룹 모두 한 개 이상의 노드가 슬라이스에 포함되면 분기노드는 실선 사각형으로 표시되며 슬라이스에 포함된다.

표 1. 예제 프로그램 2의 노드 관련 자료
Table 1. The data of related nodes for Example Program 2

[illegible]	[illegible]	[illegible]	[illegible]
1	입력	N	
2	서술	I	
3	반복		I , N
4	입력	X	
5	분기		X
6	서술	Y	X
7	서술	Y	X
8	서술	Z	Y
9	출력		Z
10	서술	I	I

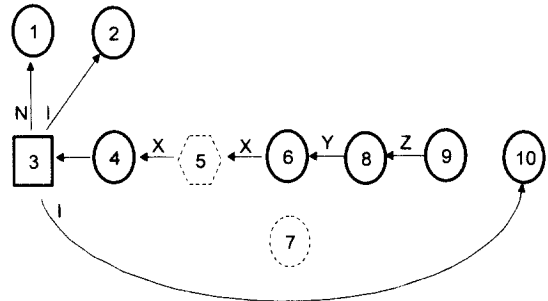



그림 4. 예제 프로그램 2의 프로그램 종속 그래프
Fig. 4. Program Dependence Graph of Example Program 2

4. 프로그램 슬라이싱 기법의 비교 및 고찰

4.1 비교 사례

하나의 시험 사례를 통한 테스트를 위해 프로그램 P의 실행 이력을 H_x 그리고 주어진 변수를 v라 한다면 H_x와 v에 관한 P의 동적 슬라이스는 어떤 실행이 실행의 종료 시점에서 관찰되어진 것처럼 v의 값에 어떤 영향을 끼치는 H_x에서의 모든 명령문들의 집합이다. 따라서 기존의 동적 프로그램 슬라이싱을 산출하기 위해서는 실행 이력을 먼저 만들어야만 한다.

본 논문에서 제안한 프로그램 슬라이싱 기법이 기존의 기법과 비교하여 실제로 프로그램 슬라이스의 크기가 줄어듦을 증명하기 위해 기존의 기법 중 대표적인 Agrawal & Horgan 기법을 선택하였다. 그리고, 비교 결과를 table로 나타내었다.

(그림 2)에 있는 예제 프로그램 2의 시험 사례로서 N = 3 이고 X = {-2, 4, -3} 일 때 프로그램 슬라이스를 구하는 경우를 Agrawal & Horgan 기법과 본 논

문에서 제안한 프로그램 슬라이싱 기법을 사용하여 서로 비교해 보았다. 그리고, 두 기법에 대한 비교 결과표가 <표 2>에 나타나 있다. Agrawal & Horgan 기법을 사용했을 때 실행 이력은 $\{1^1, 2^1, 3^1, 4^1, 5^1, 6^1, 7^1, 9^1, 10^1, 11^1, 4^2, 5^2, 6^2, 8, 9^2, 10^2, 11^2, 4^3, 5^3, 6^3, 7^2, 9^3, 10^3, 11^3, 4^4\}$가 된다. 슬라이싱 기준은 Agrawal & Horgan 기법을 사용했을 때 position 23의 변수 Z이며, 본 논문에서 제안한 기법을 사용했을 때에는 노드 10에 있는 변수 Z라고 가정한다.

표 2. 예제 프로그램 2의 슬라이스 결과 비교
Table 2. The comparison of program slicing for Example Program 2

노드 번호	Agrawal & Horgan 기법	제안한 슬라이싱 기법
1	✓	✓
2	✓	✓
3	✓	✓
4	✓	✓
5	✓	
6	✓	✓
7		
8	✓	✓
9	✓	✓
10	✓	✓

<표 2>의 결과에서 보면 8 개의 명령문으로 산출 결과가 나온 본 논문에서 제안한 기법 {1, 2, 3, 4, 6, 8, 9, 10}이 9 개의 명령문으로 산출 결과가 나온 Agrawal & Horgan 기법 {1, 2, 3, 4, 5, 6, 8, 9, 10}보다 프로그램 슬라이스의 크기가 작으므로 효율적이며 또한 실행이력이 필요 없으므로 더욱 더 실행 효율이 높음을 알 수 있다.

4.2 고찰

전통적인 동적 슬라이싱 기법의 단점은 다음과 같음을 알 수 있다. 첫째, 실행이력 파일의 산출에 따른 overhead이다. 특히 반복문에서 반복 회수가 많으면 실행이력 파일이 커지므로 이에 따른 실행효율의 저하를 가져올 수 있다. 둘째, 기준 노드를 지정할 때, 프로그램 상에서의 노드순번이 아닌 실행이력에서의 순번을 지정해야한다.

그리고, 제어 술부 노드(*while-do, if-than-else*에서의 조건 노드)는 무조건 슬라이스에 포함시키는 기존의 동적 슬라이싱 기법에 비해 본 논문에서는 노드에 직접 영향을 미치지 않는 제어 술부 노드는 제외시키는 방식이므로 제어 술부 노드라 하여도 주어진 슬라이스 기준 변수에 직접 종속되지 않으면 슬라이스에 포함시키지 않는다. 그러므로 프로그램 슬라이스의 크기가 줄어들게 되며 따라서, 기존의 슬라이싱 기법 보다 효율적임을 알 수 있다.

따라서, 프로그램에서 제어 술부 노드가 많으면 많을수록 그리고, 반복회수가 많을수록 기존 프로그램 슬라이싱 기법들에 비해 본 논문에서 제시한 프로그램 슬라이싱 기법이 더 효율적이 된다.

5. 결 론

정적 슬라이스는 주어진 기준변수에 영향을 끼치는 모든 노드를 찾는 것이고, 동적 슬라이스는 주어진 프로그램 입력에 대해 발생된 변수 값에 실제적으로 영향을 주는 명령문들로 구성되어 있다. 그러므로 어떤 시험 사례를 통해 프로그램을 분석하는 디버깅 분야에서는 동적 슬라이싱이 정적 슬라이싱 보다 더 유용하게 사용될 수 있다. 동적 스라이싱이 정적 스라이싱에 비해 정확성이 뛰어나다든지 슬라이스의 크기를 줄여준다든지 하는 여러 가지 장점을 가졌음에도 불구하고 실행이력 파일을 슬라이싱 전에 만들어야 한다는 것이 정적 슬라이싱 기법에 비해 overhead이다. 그러나 본 논문에서는 실행이력 파일을 근본적으로 만들지 않고 동적 제어 정보를 사용하여 프로그램 슬라이스를 만드는 기법을 제안하였다.

본 논문에서는 효율적인 프로그램 슬라이싱 알고리즘을 제시하였고 알고리즘을 프로그래밍 한 뒤 예제 프로그램을 적용시켜 구현하였다. 그리고, 구현 결과를 프로그램 종속 그래프로 나타내었다. 프로그램 종속 그래프에서 자료 종속 노드는 원으로 제어 종속 노드 중에서 반복 노드는 사각형으로 표시하였고, 분기 노드는 육각형으로 표시하였으며 모든 노드는 초기에 점선으로 표시되며 슬라이싱 결과에 포함된 노드는 실선으로 표현하였다.

그리고, 제안한 프로그램 슬라이싱 기법과 기존의

Agrawal & Horgan 기법을 비교하여 본 논문에서 제안한 기법이 더 효율적임을 보였다. 본 논문에서 제시한 동적 제어 정보를 이용한 프로그램 슬라이싱 기법은 제어 구문이 많을수록 특히, 반복 회수가 많은 프로그램에서 더욱 효과적임을 알 수 있었다.

참 고 문 헌

[1] Hiralal. Agrawal and J. R. Horgan. "Dynamic Program Slicing", Proc. ACM SIGPLAN'90 Conf. Programming Lang. Design and Implementaion, pp.246-256, 1990.

[2] Susan Horwitz, Jan Prins and Thomas Reps. "Integrating noninterfering versions of programs", ACM Trans. on Programming Languages and Systems, pp.345-387, July 1989.

[3] B. Korel and J. Laski, "Dynamic Slicing in Computer Programs", The Journal of System and Software Engineering, vol.23, No.1, pp.17-34, 1997.

[4] B. Korel and J. Laski, "Dynamic Program slicing", Information Proceeding Letters, vol.29, No.3, pp.155-163, 1998.

[5] Park, S. H. and M. G. Park, "An efficient dynamic program slicing algorithm and its application.", Proc. of the IASTED International Conference, Pittsburgh, Pennsylvania, pp.459-465, May 1998.

[6] 박순형, 박만곤, "소프트웨어 테스팅을 위한 동적 프로그램 슬라이싱 알고리즘의 효율성 비교", 정보처리논문지 제5권 제9호, pp.2323-2334, 1998.

[7] M. Jean and C. Ning, "Reuse-Driven Interprocedural Slicing", Proceeding of the 20th International Conference on Software Engineering, pp.74-83, 1998.04.

[8] S. Ichinose, M. Iwaihara and H. Yasuura, "Program Slicing on VHDL description and its evaluation", IEICE Transactions on Fundmentals of Electronics, vol.E81-A, No.12, pp.2585-2594, 1998.12

[9] Bodan Korel, "Computation on Dynamic Program Slices for Unstructured Programs", IEEE Trans. on Software Engineering, vol. 23, No. 1, pp.17-34, January 1997.

[10] Cheng J, "Slicing Concurrent Programs.", Proc. First Int'l Workshop Automated and Alg-orithmic Debugging, Linkoping, Sweden, pp.244-261, 1993.

[11] Korel B. and S. Yalamanchili, "Forward Derivation of Dynamic Slices.", Proc. Int'l Sy-mp. Software Testing and Analysis, Seattle, pp. 66-79, 1994.

[12] Gupta R., Harrold M. and Soffa M, "An Approach to Regression Testing Using Slicing.", Conf. Software Maintenance, pp.299-308, 1992.

[13] Robin Milner, Mads Tofte, and Robert Harper. "The Definition of Standard ML.", The MIT Press, Cambridge, MA, 1990.

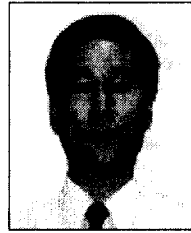
[14] Susan Horwitz, "Identifying the semantic and textual differences between two versions of a program", Proc. of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation pp.234-245, 1990.

[15] Susan Horwitz, T. Reps and David Binkley, "Interprocedural Slicing using Dependence Graph", ACM Tran. on Programming Languages and Systems, vol.12, no.1, 1990.

박 순 형

1981년 울산대학교 공과대학 전자계산학과(학사)
1985년 숭실대학교 대학원 전자계산학과(석사)
1997년 ~ 현재 부경대학교 대학원 전자계산학과 박사과정 수료
1981년 ~ 1983년 현대미포조선(주) 전산실 근무
1987년 ~ 현재 동의공업대학 전자계산과 교수
관심분야 : 소프트웨어공학 및 재공학, 소프트웨어 테스팅, 비즈니스 프로세스 재공학, 정보시스템 분석 및 설계, 비주얼 프로그래밍 기법, 멀티미디어 정보시스템 개발 방법론

정 은 이

1995년 2월 부경대학교 전자계산학과(공학사)
1997년 2월 부경대학교 대학원 전자계산학과(이학석사)
2000년 2월 부경대학교 대학원 전자계산학과(이학박사)
1997년 ~ 1999년 동명대학 전자계산과 겸임교수, 재인소프트웨어 대표.
2000년 2월 ~ 현재 춘해대학 멀티미디어정보과 교수
관심분야 : 소프트웨어공학 및 재공학, 소프트웨어 신뢰성 및 안전성 공학, 웹 기반 멀티미디어 학습정보시스템, 멀티미디어 정보시스템, 멀티미디어 소프트웨어공학 등.

박 만 곤

1976년 경북대학교 수학교육학과 졸업(이학사)
1987년 경북대학교 대학원 전산통계학과(이학박사)
1980년 ~ 1981년 경남정보대학 전자계산학과 교수
1990년 ~ 1991년 영국 리버풀대학교 전자계산학과 객원교수
1992년 ~ 1993년 미국 캔사스대학교 컴퓨터공학과 교환교수
1996년 호주 사우스 오스트레일리아대학교 컴퓨터 및 정보과학부 객원교수
1995년 몽골 컴퓨터매핑 전문가로 외무부 파견
1997년 중국 산동성 정부 정보시스템구축 전문가로 외무부에 의해서 파견
1997년 ~ 현재 콜롬보플랜 기술자교육대학(Colmbo Plan Staff College, 필리핀 마닐라에 본부) 정보기술 및 통신학처장으로 정부파견
1981년 ~ 현재 부경대학교 컴퓨터 멀티미디어 공학부 교수
관심분야 : 소프트웨어공학 및 재공학, 소프트웨어 신뢰성 및 안전성공학, 비지니스 프로세스 재공학, 멀티미디어 정보시스템, 소프트웨어품질공학, 소프트웨어 메트릭스, 소프트웨어 테스팅 및 감사, 결함허용 소프트웨어 시스템